Using K-Means to cluster users and advice venues.

Advice one restaurant or show to someone ca be a simple task. But for some reason this task can be hard. If some features of the user can be found it possible to math the correct place or show. But how make the right advice?

This notebook will try to make the correct advice of one restaurant based on similarity of users. To try this, we'll use information available on foursquare, the information collected is related about users features, likes and venues (restaurants) likes. In this way We'll try to make the right advice based on clusters of users. So we 'll collect information about users cluster them with K-Means and advice some place based on cluster of user. The steps and information used to make the advice are venues on area of interest, users like for same venues an information about these users. The steps below show this process:

A) Choose a set of venues (only restaurant) based on area of interested of user:
B) For each venue, a list of users that likes the respected venue is collected too.
C) For each user a list of features is collected.
D) Based on features of users 4 clusters are generated
E) With information of users likes one relation between cluster of users and possible venues liked is found.
F) In this way one new user can be fit on one cluster by K-Means cluster and the correct advice is obtained with venue associated with cluster.

All information is collected by foursquare API using python and the information are organized on Pandas Data Frame.

To show the results e possibility of use this approach the area chosen to start collections of venues is Manhattan - NY and latitude e longitude used is: '40.758896' and '-73.985130' respectively. To collect information of restaurant Foursquare API is used with "categoryId" variable. The partial http request is: https://api.foursquare.com/v2/venues/search?categoryId=4d4b7105d754a06374d81259&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(……..) The information provide by API results in generation of one Dataframe shown below.

	id	name	categories	address	lat	lng	Likes	UserLikeList
0	462a6065f964a520d9451fe3	Junior's Restaurant & Bakery	American Restaurant	1515 Broadway	40.758349	-73.986513	TBD	TBD
1	428a8580f964a52083231fe3	Hard Rock Cafe	Theme Restaurant	1501 Broadway	40.757035	-73.986611	TBD	TBD
2	5a8c60ef91eaca3c3c812ce6	Chick-fil-A	Fast Food Restaurant	50 E 42nd St	40.752589	-73.979205	TBD	TBD
3	4e6e89b5d22daab469b62696	Catch	Seafood Restaurant	21 9th Ave	40.740127	-74.006202	TBD	TBD
4	5cd0190f3af988002c46b0c0	Essex Market	Food Court	88 Essex St	40.718025	-73.988247	TBD	TBD
5	579c12cd498e6e815b3acf50	Eataly Downtown	Market	101 Liberty St	40.710075	-74.011976	TBD	TBD
6	598de3c33b83073f81718721	Joe's Pizza	Pizza Place	1435 Broadway	40.754679	-73.987029	TBD	TBD
7	49f85cbbf964a520f26c1fe3	Mitsuwa Marketplace	Supermarket	595 River Rd	40.816251	-73.979988	TBD	TBD
8	5748d802498eccc4228c7fd1	City Vineyard	Wine Bar	233 West St	40.721193	-74.012921	TBD	TBD
9	594277b586f4cc0f251fc389	DeKalb Market Hall	Food Court	445 Albee Sq	40.691250	-73.982579	TBD	TBD
10	4acbe67af964a52044c820e3	Katz's Delicatessen	Sandwich Place	205 E Houston St	40.722312	-73.987362	TBD	TBD
11	5467ceec498e1b95c0575e73	Pier A Harbor House	Gastropub	22 Battery Pl	40.704337	-74.018409	TBD	TBD
12	57acf9af498e6b8d5376c2d4	Westlight	New American Restaurant	111 N 12th St	40.722295	-73.956774	TBD	TBD
13	4a1467b2f964a52034781fe3	McDonald's	Fast Food Restaurant	114 Delancey St	40.718830	-73.988281	TBD	TBD
14	4a01ee7bf964a52010711fe3	P.J. Clarke's	Burger Joint	250 Vesey St	40.713358	-74.016261	TBD	TBD
15	584067e6b50d521817d726a1	Canal Street Market	Flea Market	265 Canal St	40.718890	-74.000994	TBD	TBD
16	55148c66498e987ed9d8e394	Le District	Food Court	225 Liberty St	40.712878	-74.015633	TBD	TBD
17	520e0f34498e1c964d2c9ee6	Starbucks	Coffee Shop	625 8th Ave	40.756638	-73.990947	TBD	TBD

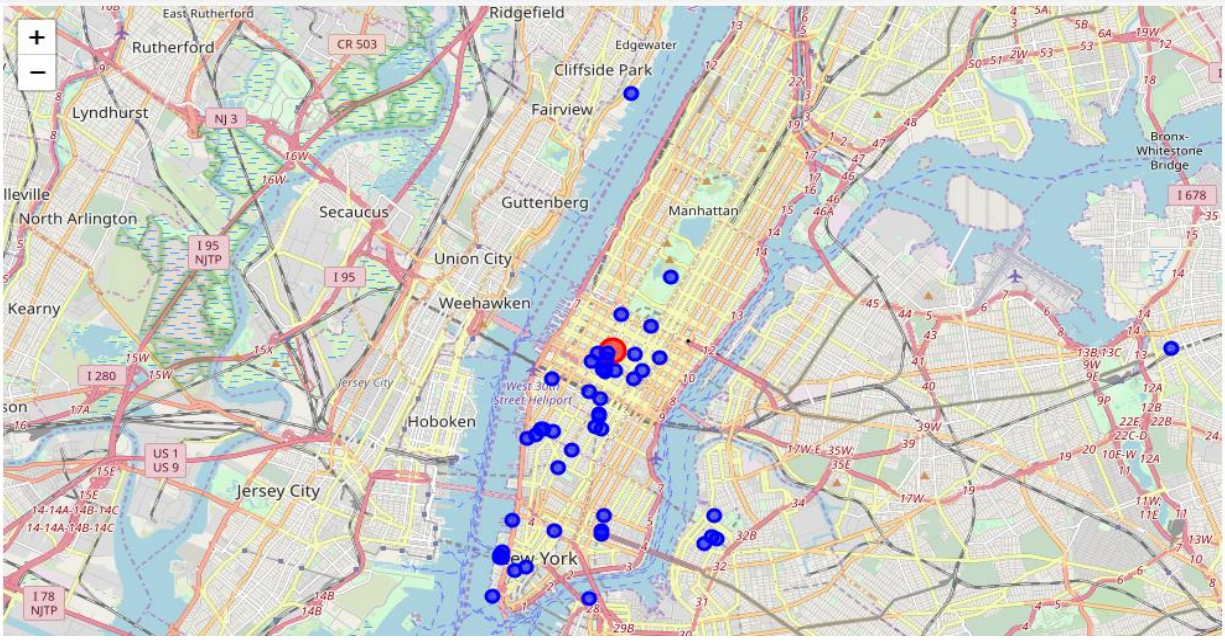
The information about users like is available only for each venue. So to a loop over each row on Dataframe to get a userlikes for the venues as shown below.

```
for index, row in nearby_venues.iterrows():
    like_list = []
    venues_likes = 'https://api.foursquare.com/v2/venues/{}/likes?&client_id={}&client_secret={}&v={}'.format(
    row['id'],
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    )
    likes = requests.get(venues_likes).json()
    for info in likes['response']['likes']['items']:
        like_list.append(info['id'])
        Users.append(info['id'])
    nearby_venues.at[index, 'UserLikeList'] = like_list
    nearby_venues.at[index, 'Likes'] = likes['response']['likes']['count']
```

After this operation the Pandas DatraFrame look like the figure below. Where is possible to see the number of likes of each venue and list of users that like the venue.

	id	name	categories	address	lat	lng	Likes	UserLikeList
0	462a6065f964a520d9451fe3	Junior's Restaurant & Bakery	American Restaurant	1515 Broadway	40.758349	-73.986513	1788	[368000, 16189616, 453809652]
1	428a8580f964a52083231fe3	Hard Rock Cafe	Theme Restaurant	1501 Broadway	40.757035	-73.986611	2951	[409389877, 102083859, 551882420]
2	5a8c60ef91eaca3c3c812ce6	Chick-fil-A	Fast Food Restaurant	50 E 42nd St	40.752589	-73.979205	41	[146073114, 4160, 400223414]
3	4e6e89b5d22daab469b62696	Catch	Seafood Restaurant	21 9th Ave	40.740127	-74.006202	1104	[511343881, 394347798]
4	5cd0190f3af988002c46b0c0	Essex Market	Food Court	88 Essex St	40.718025	-73.988247	54	[3084086, 410193106, 517314948]
5	579c12cd498e6e815b3acf50	Eataly Downtown	Market	101 Liberty St	40.710075	-74.011976	1143	[152489174, 2332145, 58561787]
6	598de3c33b83073f81718721	Joe's Pizza	Pizza Place	1435 Broadway	40.754679	-73.987029	289	[349949235, 6228033, 226293226]

The figure below uses folium to show the positions of venues on map.



The same map cannot be printout for user because we have user for different places, cities and even countries.

Now we need to uses API to collect information for each user. These information's will be used to create a cluster of users. Is based on cluster or users that advice is evaluated. The figures below show the API request and padas DataFrame as result of these requests.

```
for user in Users:
    inner_user_dict = {}
    url_users = 'https://api.foursquare.com/v2/users/{}?&client_id={}&client_secret={}&v={}'.format(
    user,
```

```
<bound method DataFrame.head of             ID checkins  gender                                homecity  \
0       368000    10201  female                                New York
1     16189616      968  female                           Cotia, Brasil
2    453809652        1    male
3    409389877      466  female
4    102083859       19    male                             Ufa, Russia
5    551882420       14    none                             Chicago, IL
6    146073114      683    male                  New York, New York
7         4160     4447    male                Poughkeepsie, New York
8    400223414      155    none                             New York, NY
9    511343881      726  female
10   394347798      664  female
11     3084086     5807    male                          Parsippany, NJ
12   410193106      381  female                            Brooklyn, NY
13   517314948        0    male
14   152489174     2200    male  Aartrijke, West-Vlaanderen, Belgium
15     2332145     3746  female                             New York, NY
16    58561787     3707    none                                Singapore
17   349949235       34  female                              Warsaw, 78
```

As come information arrive as strings, come transformations is needed to perfect fits on K-Means algorithm, as shown below.

```
<bound method DataFrame.head of             ID checkins  gender  homecity mayorships n_friends n_photos  \
0       368000    10201       1        59          0        48        0
1     16189616      968       1        17          0        41        0
2    453809652        1       2         0          0         0        0
3    409389877      466       1         0          0         0        0
4    102083859       19       2        14          0         0        0
5    551882420       14       3        55          0         0        0
6    146073114      683       2        12          0        16        0
7         4160     4447       2        40          0        61        0
8    400223414      155       3         8          0         0        0
9    511343881      726       1         0          0         0        0
10   394347798      664       1         0          0         0        0
11     3084086     5807       2        15          0        20        0
12   410193106      381       1        50          0         0        0
13   517314948        0       2         0          0         0        0
14   152489174     2200       2        58          0        52        0
15     2332145     3746       1         8          0        43        0
16    58561787     3707       3        21          0         2        0
17   349949235       34       1        39          0         0        0
```

Now these information's are used to create 4 groups of users with help of K-Means.

```
#Now we can use KMeans to clusters uses
#Here we cluester users on 4 clusteres
UsersDF2 = UsersDF.drop(columns=['ID'])
km = KMeans(n_clusters = 4)
km.fit(UsersDF2.values)
prediction = km.predict(UsersDF2.values)
UsersDF['cluster'] = prediction
```

The result of clustering algorithms is one dataframe where each user belongs to only on of 4 groups.

	ID	checkins	gender	homecity	mayorships	n_friends	n_photos	superuser	tips	type	cluster
0	368000	10185	1	56	0	48	0	1	52	1	2
1	16189616	963	1	19	0	41	0	0	22	1	1
2	453809652	1	2	0	0	0	0	0	0	1	1
3	551882420	14	3	52	0	0	0	0	0	1	1
4	152489174	2172	2	55	0	52	0	0	36	1	3
5	9014241	4460	1	18	0	0	0	0	1	1	3
6	146073114	683	2	14	0	16	0	0	1	1	1
7	4160	4444	2	39	0	61	0	0	3	1	3
8	400223414	155	3	10	0	0	0	0	48	1	1
9	511343881	720	1	0	0	0	0	0	1	1	1
10	394347798	662	1	0	0	0	0	0	0	1	1
11	65890	15763	2	10	0	311	0	0	122	1	0
12	3084086	5806	2	17	0	20	0	0	55	1	2
13	410193106	378	1	48	0	0	0	0	0	1	1
14	517314948	0	2	0	0	0	0	0	7	1	1
15	152489174	2172	2	55	0	52	0	0	36	1	3
16	2332145	3739	1	10	0	43	0	0	4	1	3
17	58561787	3681	3	22	0	2	0	0	66	1	3

The last step is relation each group (cluster) to best venue. Because with this relation news users can be classifier to one cluster and then advice to one restaurant. The relation between cluster and venues I archive with the sum o most venue for each user on cluster, as shown below.

```
#Whith each cluster user with all venues liked by uses on same cluester
#We can adviser the most probabily venue for user on same cluster.
venues_vs_user_group = {}
for key,value in user_group.items():
    group_nearby_venues_list = []
    #print('Group',key,'Users:',value)
    for users_id in value:
        #print('testing user',users_id)
        for index, row in nearby_venues.iterrows():
            #print('testing row',index)
            like_list_to_test = row['UserLikeList']
            if users_id in like_list_to_test:
                group_nearby_venues_list.append(row['id'])
                #print('Matching for:',row['id'])

    #print('-->',group_nearby_venues_list,len(group_nearby_venues_list),len(set(group_nearby_venues_list)))
    venues_vs_user_group[key] = {'user_count':len(value),'users':value,'venues_count':len(set(group_nearby_venues_list)),
                                 'top_venue':  max(set(group_nearby_venues_list), key = group_nearby_venues_list.count) }
    print(venues_vs_user_group[key])
    print('-----------------------------------------------------------------------------------')
```

Now each group of users (cluster) have one best venue for advice…


```
{'venues_count': 9, 'user_count': 9, 'top_venue': u'5362a2ae498e3b18c22334be', 'users': [u'65890', u'13839018', u'3582397', u'7
444795', u'2006845', u'71206817', u'1491653', u'367159', u'7447793']}
--------------------------------------------------------------------------------
{'venues_count': 44, 'user_count': 71, 'top_venue': u'5467ceec498e1b95c0575e73', 'users': [u'16189616', u'453809652', u'5518824
20', u'146073114', u'400223414', u'511343881', u'394347798', u'410193106', u'517314948', u'226293226', u'501320653', u'49442173
0', u'200826104', u'392376955', u'93943139', u'421280744', u'82677334', u'63996744', u'522280923', u'492037624', u'384468769',
u'525854205', u'50201177', u'506876345', u'22691', u'60484468', u'505590360', u'487871986', u'49496611', u'460948204', u'479794
342', u'17859217', u'35040924', u'533947886', u'45144401', u'3484561', u'34858101', u'498921705', u'4868604', u'82677334', u'19
032000', u'118871833', u'555211531', u'441712', u'492037624', u'126784566', u'514117249', u'373844596', u'88107401', u'4270926
7', u'556956131', u'20910655', u'1563509', u'121781647', u'556004000', u'492037624', u'88344930', u'52277259', u'135408382',
u'38320704', u'23551997', u'82441683', u'12644111', u'509971477', u'92994549', u'524632668', u'480944400', u'27244950', u'40945
2259', u'72131520', u'357992775']}
--------------------------------------------------------------------------------
{'venues_count': 17, 'user_count': 20, 'top_venue': u'5716655d498e23619f894ef0', 'users': [u'368000', u'3084086', u'55965888',
u'24767630', u'2651498', u'3884079', u'323932', u'10658105', u'1482283', u'1505916', u'13791', u'62167634', u'72391463', u'1048
5250', u'2717634', u'2717634', u'60726', u'51697751', u'5767928', u'479524']}
--------------------------------------------------------------------------------
{'venues_count': 31, 'user_count': 42, 'top_venue': u'579c12cd498e6e815b3acf50', 'users': [u'152489174', u'9014241', u'4160',
u'152489174', u'2332145', u'58561787', u'6228033', u'6532455', u'11259584', u'73230983', u'17672837', u'148715595', u'6887848',
u'16535617', u'8243899', u'6532455', u'75806155', u'46726215', u'54372', u'37929812', u'31109863', u'5476237', u'7658576', u'18
451425', u'152489174', u'118194432', u'54430683', u'73230983', u'140206438', u'196963253', u'153786', u'4522362', u'59677304',
u'46435929', u'31585196', u'1971638', u'25694261', u'1718190', u'18480013', u'4157112', u'51177', u'196963253']}
--------------------------------------------------------------------------------
```


News users can be classifier and placed on one cluster, consequently will receive the advice for venue of this cluster. In this way the new user receives the advice by similarity to the other users belonging to the cluster in which it was classified.